EECS C145B / BioE C165: Image Processing and Reconstruction Tomography

Lecture 11 (revision II)

Jonathan S. Maltz

jon@eecs.berkeley.edu
http://muti.lbl.gov/145b
510-486-6744

Topics to be covered

1. Introduction to iterative methods
2. Introduction to the algebraic reconstruction technique (ART) (See tomographic reconstruction primer)
3. Modeling image statistics: The Gaussian distribution
4. The maximum likelihood (ML) principle
5. Relationship between Gaussian ML and least squares
6. Modeling image statistics: The Poisson process
7. Introduction to optimization
8. Solving ML problems using optimization algorithms

Reading

Assigned reading:

- Budinger, "A Primer on Reconstruction Algorithms" (handout).

Optional reading

- Stirzaker, "Elementary Probability", Cambridge University Press (1994), pp. 290-293.
- Lawler, "Introduction to Stochastic Processes", Chapman & Hall (1995), pp. 52-56.
- "Numerical Recipes in C". Press, Teukolsky, Vetterling and Flannery, 2nd Edition (1995) pp. 656-666.

Advanced reading:

- Natterer, "The Mathematics of Computerized Tomography", John Wiley and Sons (1986), Chapter V.

Introduction to iterative algorithms

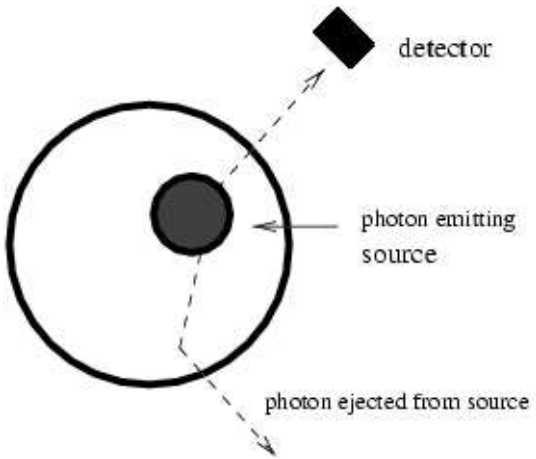
- The pseudoinverse method of tomographic image reconstruction **is too slow and requires too much memory** for the solution of even moderately sized problems. This is because it tries to solve all the linear equations that relate the pixel values to the projection measurements **in one step**.
- The analytical reconstruction methods we derived from the projection slice theorem (PST) are currently the **most popular** algorithms for reconstructing tomographic images.
- These algorithms are **very fast** and **easy to implement**.
- Unfortunately, within these algorithms it is **not possible to accurately model** many of the real-world processes that degrade the quality of images. These include:

Introduction to iterative algorithms

1. **Photon scatter:** In emission tomography, the **interaction of an emitted gamma photon with matter** on its way out of the body changes the **energy and direction of that photon.** Photons generated at a source lying along one ray thus arrive at the detector along another ray. This leads to errors in the projection data that increase noise and blur in the reconstructed image.
2. **Attenuation:** In emission tomography, photons generated within the body **have to travel through attenuating matter** before these photons reach the detector. Thus, emitted photons that have to travel further through high density material will have a lower likelihood of arriving at the detector than photons emitted closer to the detector. An attenuation map may be created using an x-ray CT scan so that attenuation may be compensated for. However, algorithms based on the projection theorem cannot utilize this information.

Introduction to iterative algorithms

3. **Photon statistics:** We will see shortly that the PST-based methods and the pseudoinverse method assume that each projection bin is a realization of a **Gaussian distribution with unit variance**. In reality, photon counts follow a **Poisson distribution**.



Photons are emitted from a radioactive source within a body. The lower photon is **scattered** on its way out and appears to come from a source along a different ray.

Modeling image statistics

- We will first investigate methods of modeling photon statistics. Later in the course, we will discuss scatter and attenuation.
- We begin by modeling the number of photons detected in a single projection bin as being distributed according to a **Gaussian distribution.**
- We model the number of counts measured in the jth projection bin of the ith angular projection as:

$$p_{ij} = \lambda_{ij} + \epsilon$$

  where λ_{ij} is the true number of counts, and ϵ is some noise that has contaminated the measurement.
- We can model ϵ as a Gaussian random variable that has a mean of zero, and a variance of σ_ϵ^2.

Modeling image statistics: Gaussian noise model

- The Gaussian **probability density function** (pdf) tells us the probability that a certain value ϵ of the random variable (rv) E will occur:

$$f_E(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_\epsilon} \, \mathrm{e}^{-\frac{1}{2\sigma_\epsilon^2}\epsilon^2},$$

- The Gaussian (or any other) **cumulative distribution function** (cdf) gives us the probability that the rv E is less than a given value ϵ:

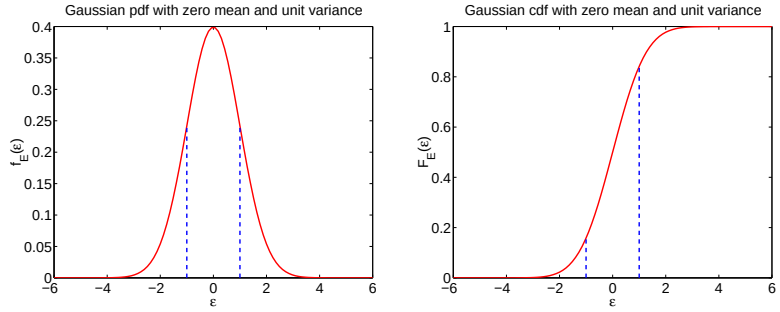
$$F_E(\epsilon) = P(\epsilon \leq E) = \int_{-\infty}^{\epsilon} f_E(\omega)\, d\omega$$

- We can also use the cdf to find the probability that E will fall within a certain range:

$$P(\epsilon_1 < E \leq \epsilon_2) = \int_{\epsilon_1}^{\epsilon_2} f_E(\omega)\, d\omega$$

Modeling image statistics: Gaussian noise model

- For example, plotted below are the Gaussian pdf and cdf for a random variable E that has a mean of zero and variance of one.



- Note that **the total area under any pdf must be unity**. As a consequence, **the final value of any cdf must be unity.**
- Imagine a hat full of an infinite number of pieces of paper, each with a value ϵ. The numbers of on the pieces of paper follow a Gaussian distribution with zero mean and unit variance.

Modeling image statistics: Gaussian noise model

1. What value are you most likely to draw from the hat?

---

2. What is the probability that the value ϵ on a piece of paper you have drawn falls within one standard deviation of the mean?

$$\begin{aligned} P(-1 < E \leq 1) &= \int_{-1}^{1} f_E(\omega)\, d\omega \\ &= \int_{-\infty}^{1} f_E(\omega)\, d\omega - \int_{-\infty}^{-1} f_E(\omega)\, d\omega \\ &= F_E(1) - F_E(-1) \\ &= 0.8413 - 0.1587 = 0.6827 \end{aligned}$$

So, 68% of the time we will draw a number within one standard deviation of the mean.

3. How probable is it that you draw the most likely value?

Modeling image statistics: Gaussian noise model

4. What is the probability that the value ϵ on a piece of paper you have drawn falls within two standard deviations of the mean?

$$\begin{aligned} P(-2 < E \leq 2) &= \int_{-2}^{2} f_E(\omega)\, d\omega \\ &= \int_{-\infty}^{2} f_E(\omega)\, d\omega - \int_{-\infty}^{-2} f_E(\omega)\, d\omega \\ &= F_E(2) - F_E(-2) \\ &= 0.9772 - 0.8413 = 0.9545 \end{aligned}$$

So, over 95% of the area under the Gaussian pdf lies within two standard deviations of the mean.

5. Suppose we have two Gaussian rv's D and E and both have the same pdf and are independent. What is the probability that $D \leq 0$ and $E \leq 0$?

Modeling image statistics: Gaussian noise model

Well, we know that D has equal probability of being $\geq$ than zero and $\leq$ zero. Therefore:

$$P(D \leq 0) = 0.5$$

Similarly

$$P(E \leq 0) = 0.5.$$

But what are the chances that both are less than or equal to zero? We have four possible scenarios:

$$D \leq 0 \quad \text{and} \quad E \leq 0$$
$$D \leq 0 \quad \text{and} \quad E > 0$$
$$D > 0 \quad \text{and} \quad E \leq 0$$
$$D > 0 \quad \text{and} \quad E > 0$$

Therefore, only one in four $\left(\frac{1}{2} \times \frac{1}{2}\right)$ times will both rvs be less than or equal to zero.

Modeling image statistics: Gaussian noise model

In general, for two independent events A and B

$$P(A \cap B) = P(A) \times P(B)$$

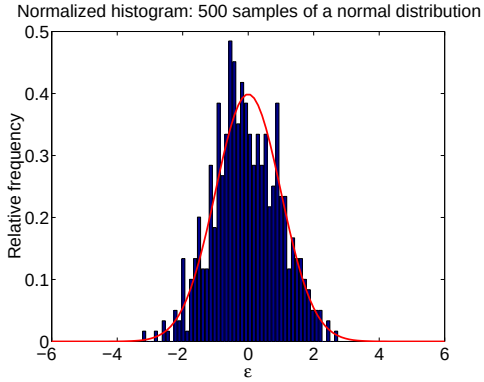
For N indepedent events A_1 through A_N,

$$P(\cap_{i=1}^{N} A_i) = \prod_{i=1}^{N} P(A_i)$$

This makes intuitive sense. The chance of two events occurring simultaneously will always be less than or equal to the probability of just one of the events occurring. All probabilities are in the interval $[0, 1]$. Multiplying a fraction by a fraction will always give a smaller fraction.
Example: When we throw a fair die, we have a chance of $1/6$ of getting a six. When we throw two dice, our chance of getting two sixes is much less likely. In fact, it is $1/6 \times 1/6 = 1/36$.

Modeling image statistics: Gaussian noise model

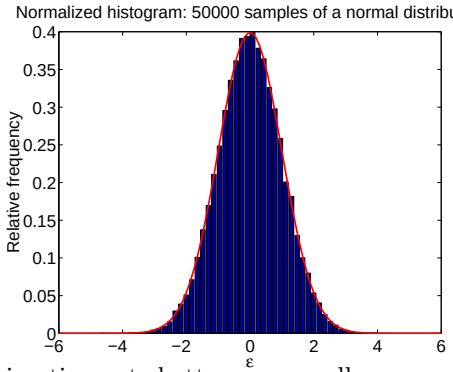
4. Returning to the subject of drawing ϵ's out of the "hat", what will we get if we draw 500 numbers out of the hat and then plot a histogram of these numbers?



We see that the histogram approximates the Gaussian pdf. The histogram gives us the empirical probablility (or **relative frequency**) of the value of E falling within the limits of each of the 50 histogram bins.

Modeling image statistics: Gaussian noise model

6. What will we get if we draw 50000 numbers out of a hat and plot a histogram of these numbers?



The approximation gets better as we pull more numbers out of the hat. When we draw an **infinite number** of pieces of paper, **our histogram becomes the pdf. Relative frequency** becomes **probability.**

Modeling image statistics: Gaussian noise model

So, what does this have to do with imaging?

- Imagine that the number of photons that hit a detector in a projection bins in a time Δt is equivalent to pulling a number ϵ out of a hat and adding it to an unknown "true measurement" λ_{ij}. As before:

$$f_E(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_\epsilon} \, \mathrm{e}^{-\frac{1}{2\sigma_\epsilon^2}\epsilon^2},$$

and

$$p_{ij} = \lambda_{ij} + \epsilon$$

- For an entire set of projections, the probability of getting a certain set of ϵ's is the product of the probabilities of getting a certain ϵ for a single bin:

$$f_{\mathbf{E}}(\boldsymbol{\epsilon}) = \prod_{i=1}^{I} \prod_{j=1}^{J} f_{Eij}(\epsilon_{ij})$$

Modeling image statistics: Gaussian noise model

- This expression represents the product of IJ pdfs. For independent Gaussian distributions, this product is the joint Gaussian distribution:

$$f_{\mathbf{E}}(\boldsymbol{\epsilon}) = \frac{1}{(2\pi)^{IJ/2} \prod_{i=1}^{I} \prod_{j=1}^{J} \sigma_{ij}} \, \mathrm{e}^{-\frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J}\frac{1}{\sigma_{ij}^2}\epsilon_{ij}^2}$$

where σ_{ij}^2 is the variance of ϵ_{ij}.

- Subsitituting $p_{ij} - \lambda_{ij}$ for ϵ_{ij}, we get:

$$f_{\mathcal{P}|\Lambda}(\mathbf{p}, \boldsymbol{\lambda}) = \frac{1}{(2\pi)^{IJ/2} \prod_{i=1}^{I} \prod_{j=1}^{J} \sigma_{ij}} \, \mathrm{e}^{-\frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J}\frac{1}{\sigma_{ij}^2}(p_{ij}-\lambda_{ij})^2}$$

Here, the vector $\boldsymbol{\lambda}$ contains the λ_{ij}. This pdf gives us the probability that our projection measurements $\mathbf{p}$ came from the projections of our image $\boldsymbol{\lambda} = \mathbf{F}\boldsymbol{\mu}$. In other words, it describes the probability of the measurements $\mathbf{p}$ **given** the model's prediction $\boldsymbol{\lambda}$ ($\mathbf{F}$ is the discrete Radon transform matrix).

Modeling image statistics: Gaussian noise model

- We now construct an $IJ \times IJ$ diagonal matrix $\boldsymbol{\Sigma}$ that contains the variances σ_{ij}^2 on its diagonal. Then we can express this joint pdf more compactly in vector form as:

$$f_E(\boldsymbol{\epsilon}) = \frac{1}{(2\pi)^{IJ/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \, \mathrm{e}^{-\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\epsilon}}$$

or

$$f_{\mathcal{P}|\Lambda}(\mathbf{p}, \boldsymbol{\lambda}) = \frac{1}{(2\pi)^{IJ/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \, \mathrm{e}^{-\frac{1}{2}(\mathbf{p}-\boldsymbol{\lambda})^T\boldsymbol{\Sigma}^{-1}(\mathbf{p}-\boldsymbol{\lambda})}$$

or

$$f_{\mathcal{P}|\mathcal{U}}(\mathbf{p}, \boldsymbol{\mu}) = \frac{1}{(2\pi)^{IJ/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \, \mathrm{e}^{-\frac{1}{2}(\mathbf{p}-\mathbf{F}\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{p}-\mathbf{F}\boldsymbol{\mu})}$$

where $|\cdot|$ denotes the determinant.

Modeling image statistics: Gaussian noise model

- The exponent in the function $f_{\mathcal{P}|\mathcal{U}}(\mathbf{p}, \boldsymbol{\mu})$ is always negative, because:
  1. The inverses of the variances σ_{ij}^2, which make up the diagonal of $\boldsymbol{\Sigma}^{-1}$, are all non-negative.
  2. The quadratic form $(\mathbf{p} - \mathbf{F}\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{p} - \mathbf{F}\boldsymbol{\mu})$ is always non-negative when $\boldsymbol{\Sigma}^{-1}$ has only non-negative elements.
- Since the exponent of the exponential factor in $f_{\mathcal{P}|\mathcal{U}}(\mathbf{p}, \boldsymbol{\mu})$ is always non-positive, the maximum value of this factor is one. This occurs when $\mathbf{p} = \mathbf{F}\boldsymbol{\mu}$. This is the case when the projections of the image defined by $\boldsymbol{\mu}$ exactly match the projection measurements.

Maximum likelihood

- By inspection, we have just solved the Gaussian **maximum likelihood (ML)** problem. We found the solution that **maximizes the probability of getting the measurements we obtained** ($\mathbf{p}$), **given the model we have assumed** ($\mathbf{p} = \mathbf{F}\boldsymbol{\mu}$).
- Generally, we can't solve problems ML problems by inspection. We normally solve them by **minimizing** the **negative of the natural log of the pdf describing the probability of getting the measurements given the model**. This expression is treated as a function of the parameters $\boldsymbol{\mu}$ alone. For the joint Gaussian distribution, the negative log-likelihood function is:

$$\ell(\boldsymbol{\mu}) \quad = \quad -\ln\left[\frac{1}{(2\pi)^{IJ/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\right] + \frac{1}{2}(\mathbf{p}-\mathbf{F}\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{p}-\mathbf{F}\boldsymbol{\mu})$$

Maximum likelihood: Gaussian distributions

- This can be expanded as:

$$\ell(\boldsymbol{\mu}) \quad = \quad -\ln\left[\frac{1}{(2\pi)^{IJ/2}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\right] + \frac{1}{2}\mathbf{p}^T\boldsymbol{\Sigma}^{-1}\mathbf{p} - 2\,\mathbf{p}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}\boldsymbol{\mu} + \boldsymbol{\mu}^T\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}\boldsymbol{\mu}$$

- To minimize this function, we take its derivative with respect to the parameters $\boldsymbol{\mu}$ and set this equal to zero:

$$\frac{d\ell(\boldsymbol{\mu})}{d\boldsymbol{\mu}} = \frac{1}{2}\Big[-2\,\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{p} + 2\,\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}\boldsymbol{\mu}\Big] = 0$$

- Solving for $\boldsymbol{\mu}$ gives:

$$\boldsymbol{\mu} = \left(\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{p}$$

Maximum likelihood: Gaussian distributions

- Note that when we assign unit variance to each of the μ, $\boldsymbol{\Sigma}$ becomes the identity matrix and:

$$\boldsymbol{\mu} = \boldsymbol{\mu}_{\text{LS}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{p} = \mathbf{F}^+\mathbf{p}$$

  This is the pseudoinverse we derived earlier by minimizing the sum of squared residuals.

- Viewed from the standpoint of statistical parameter estimation, the pseudoinverse finds that solution that maximizes the likelihood of the data given the model **assuming that the data come from a Gaussian process where each datum is an independent rv having unit variance.**

Maximum likelihood: Gaussian distributions

- Since we know that the emission of photons from the nuclei of atoms is better modeled as an independent Poisson process, the Gaussian statistical model is **not optimal.**
- The **weighted least squares (WLS)** formulation:

$$\boldsymbol{\mu}_{\text{WLS}} = \left(\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^T\boldsymbol{\Sigma}^{-1}\mathbf{p}$$

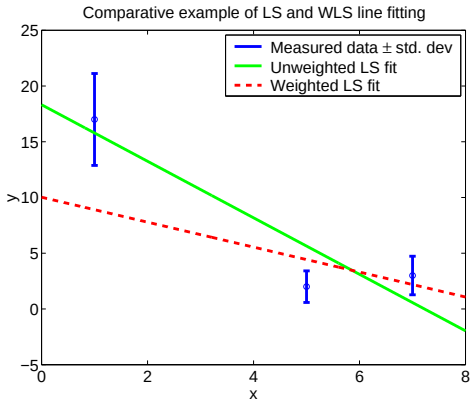
  is more general, and better approximates reality, since we can specify the variance of each data point independently. Consequently, measurements in which we have more confidence are more highly **weighted** in the cost function than those in which we have lower confidence.

- In WLS, we minimize the **sum of the squares of the weighted residuals:** (compare to the cost function in Lecture 8)

$$C = \sum_{m=1}^{M} r_m^2 = \sum_{m=1}^{M}\left[\sum_{n=1}^{N} f_n^m(x_m)\,\theta_n - y_m\right]^2/\sigma_m^2$$

Weighted least squares line fitting example

Suppose we obtain some physical measurements along with the calculated uncertainties of these measurements (standard deviations). Instead of weighting the residuals of each point equally, we weight each by the inverse of the variance of each:



We see that WLS "pulls" the line towards the measurements that have lower variance. This makes much more sense than assuming all measurements have the same variance. Low variance points "pull" the line with stronger "rubber bands".

The Poisson process

- We now investigate the statistical modeling of radioactive decay in more detail so we can choose a maximum likelihood approach that better matches physical reality.
- A nucleus of an atom decays, releasing a photon, independently of what is going on in other nuclei. As a result, knowing the time at which the last photon was emitted from a material **tells us nothing** about when the next photon will be emitted.
- If we assume that the **average number** of photons emitted during a time interval is **constant**, and if we assume that photons are emitted **one at a time**, we can model radioactive decay as a **Poisson process**.

The Poisson process

More formally: Consider $N(t)$ to be the total number of photons that have arrived at a detector by time t. We assume that:

1. The number of photons arriving during one time interval does not affect the number arriving during a different time interval.
2. The mean rate of photon arrival (β) remains constant (The half-life of the radioactive material is much longer than the time interval $[0, t]$).
3. Photons arrive one at a time.

Let $\Delta T_n; n \geq 1$ be the interval between the arrival of the $(n-1)$st and nth photons. Then

$$T_n = \sum_{k=1}^{n} \Delta T_k$$

is the total amount of time unit n photons have arrived. We can write:

$$\Delta T_n = T_n - T_{n-1}$$

The Poisson process

In order to satisfy assumption (1), the rv ΔT_n must satisfy the "loss of memory" property. In other words, if we have waited for a time period of s time units and no photon has arrived, the chance of one arriving during the next t time units is exactly the same as it would have been if some photons **had** arrived while we were waiting. Formally:

$$P(\Delta T_i \geq s + t \,|\, \Delta T_i \geq s) = P(\Delta T_i \geq t)$$

"The probability that we will wait more than $s + t$ time units for photon to be detected given that we have already waited s time units, is the same as the probability that a photon will be detected in the next t time units."

The Poisson process

- General probability theory tells us that:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

- The intersection event can be evaluated as:

$$P(\Delta T_i \geq s + t \,|\, \Delta T_i \geq s) = P(\Delta T_i \geq s + t)$$

giving:

$$P(\Delta T_i \geq s + t) = P(\Delta T_i \geq s)\, P(\Delta T_i \geq t)$$

- The only real-value funtions for which:

$$f(t + s) = f(t)\, f(s)$$

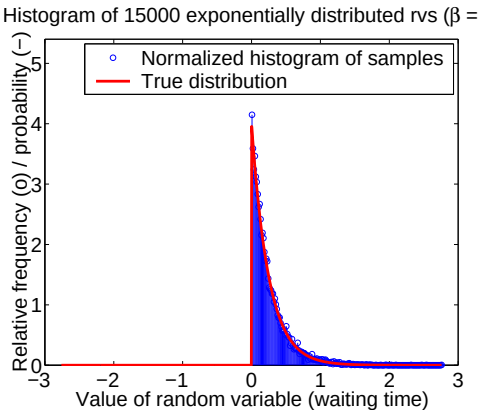
are exponentials of the form:

$$f(t) = k\mathrm{e}^{-\beta t}$$

- Intuitively, this is because we can cut any real decaying exponential at any non-negative time point, take the part of the function to the right of the cut point and be assured that this function will always be another real decaying exponential function.

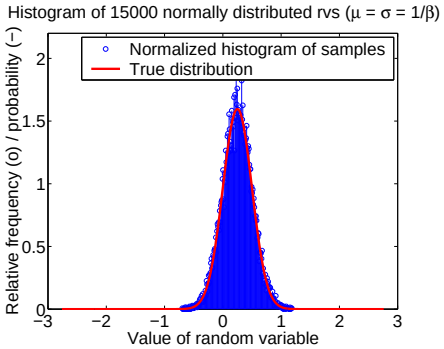
The Poisson process

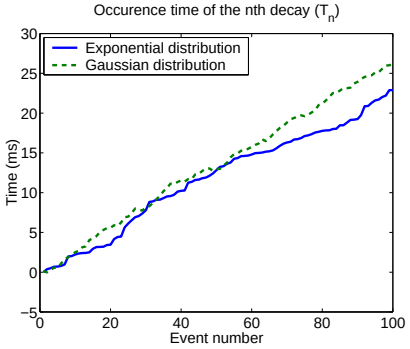
- **The waiting times between the arrival of photons follow an exponential distribution**. The process in which these times are distributed in this way is called a **Poisson process**.



The exponential distribution $p(t) = \beta\, \mathrm{e}^{-\beta t}$ describes the probability of waiting a period of t time units for the next photon to arrive.

The Poisson process





Here, we compare the occurrence time of the nth events for the Poisson process (with exponentially distributed intervals) to a process that has intervals sampled from a Gaussian distribution with the same mean and variance as the exponential distribution. Note how the left tail of the Gaussian can cause the $(n+1)$st to event occur before the nth event.

The Poisson process

Properties of the exponential distribution:

1. Mean $= \frac{1}{\beta}$. This is the mean waiting time between events in the Poisson process.
2. Variance $= \frac{1}{\beta^2}$.

β has units of inverse time, and so describes the mean rate of photon arrival.

The Poisson process

- For the Poisson process, the number of events that occur in a time interval is proportional to the length of the time interval, and the **rate parameter** β and **nothing else**.
- It can be shown that the probability that k photons have arrived by time t is given by:

$$P(N(t) = k) = \mathrm{e}^{-\beta t} \frac{(\beta t)^k}{k!}$$

- The general Poisson probability mass function (PMF) is defined as:

$$P(N = k) = \mathrm{e}^{-\lambda} \frac{(\lambda)^k}{k!}$$

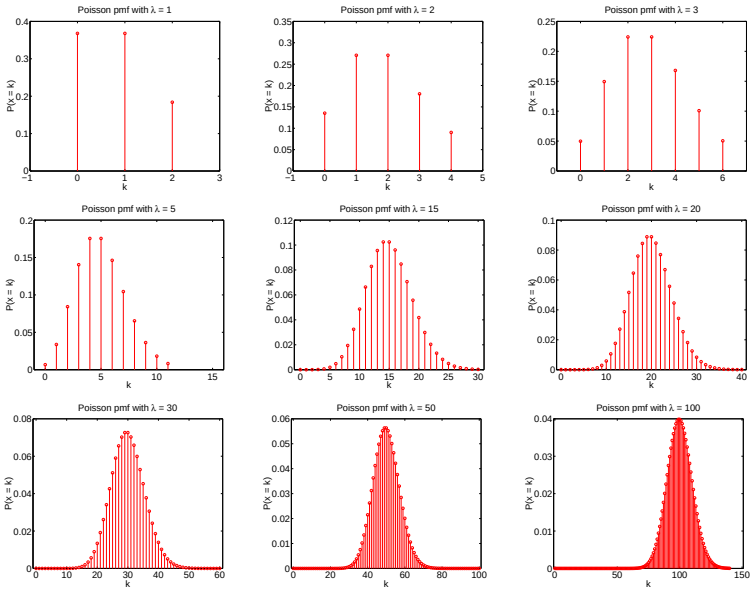
Thus, the Poisson process $N(t)$ has a **Poisson distribution** with parameter $\lambda = \beta t$.

The Poisson process

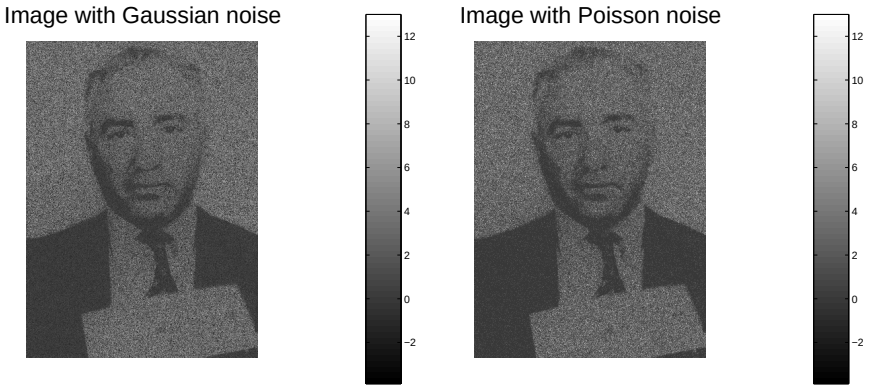
Properties of the Poisson distribution:

1. The mean of the Poisson distribution is λ.
2. The variance of the Poisson distribution is λ.
3. The sum of independent Poisson random variables is also Poisson and has mean equal to the sum of the means of its components.
4. As λ increases, the Poisson distribution becomes a better and better discrete approximation to the Gaussian distribution with $\mu = \lambda$ and $\sigma = \sqrt{\lambda}$.

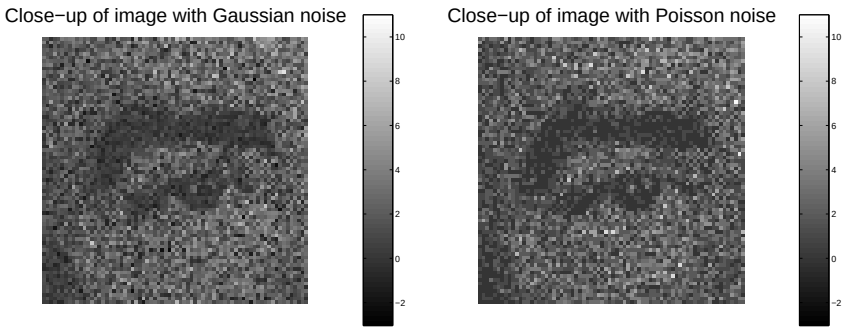
The Poisson distribution



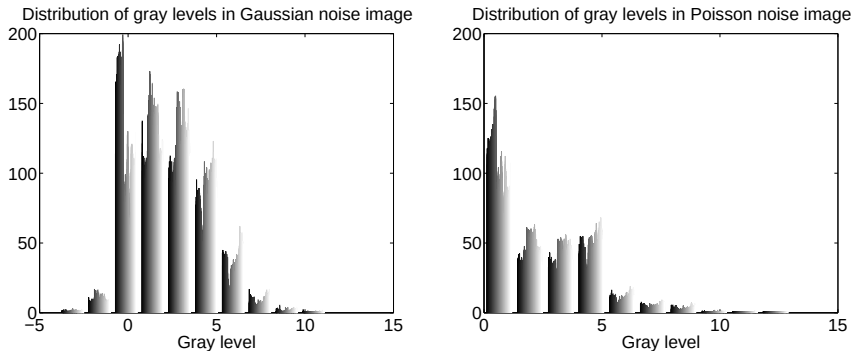
The Poisson distribution



The Poisson distribution



The Poisson distribution



Maximum likelihood for Poisson distributions

- Recall that the Poisson process is defined by the pmf:

$$P(N(t) = k) = \mathrm{e}^{-\beta t} \frac{(\beta t)^k}{k!}$$

- Imagine that $N(t)$ is the number of photons detected within a projection bin after a time t. If we take $t = T$ seconds to record a single projection, our model tells us to expect $\lambda = \beta t$ counts to be detected.
- The probability of recording $\lambda_{ij} = p_{ij}$ counts in the jth bin of the ith angular projections during the total imaging period is:

$$P(\lambda_{ij} = p_{ij}) = \mathrm{e}^{-\lambda_{ij}} \frac{\lambda^{p_{ij}}}{p_{ij}!}$$

Maximum likelihood for Poisson distributions

- As in the Gaussian case, we assume all bins are independent. Thus:

$$P(\boldsymbol{\lambda} = \mathbf{p}) = \prod_{i=1}^{I} \prod_{j=1}^{J} P(\lambda_{ij} = p_{ij})$$

- We now wish to maximize the likelihood that this model $\boldsymbol{\lambda} = \mathbf{F}\boldsymbol{\mu}$ created the data $\mathbf{p}$.
- To do this, we form the negative log-likelihood function:

$$\ell(\boldsymbol{\mu}) \quad = \quad -\sum_{i=1}^{I} \sum_{j=1}^{J} -\lambda_{ij}(\boldsymbol{\mu}) + p_{ij} \ln(\lambda_{ij}(\boldsymbol{\mu})) - \ln(p_{ij}!)$$

where the p_{ij} are the projection bin measurements (elements of $\mathbf{p}$) and the λ_{ij} are the elements of the Poisson process parameter vector $\boldsymbol{\lambda} = \mathbf{F}\boldsymbol{\mu}$.

Maximum likelihood for Poisson distributions

- To maximize the likelihood, we minimize the negative log likelihood by taking its derivatives with respect to the μ_n and setting these N equations equal to zero:

$$\frac{d\ell(\boldsymbol{\mu})}{d\mu_n} = -\sum_{i=1}^{I}\sum_{j=1}^{J} -\frac{d\lambda_{ij}(\boldsymbol{\mu})}{d\mu_n} + \frac{p_{ij}}{\lambda_{ij}(\boldsymbol{\mu})}\frac{d\lambda_{ij}(\boldsymbol{\mu})}{d\mu_n} = 0 \qquad n = 1, 2, \ldots, N$$

- When we maximized the Gaussian likelihood, these equations were linear in $\boldsymbol{\mu}$. For the Poisson distribution, they are **non-linear**. We cannot, in general, find a closed-form solution to this problem (such as a pseudoinverse).
- We must use iterative methods to find the ML estimate. We must also ensure that all the $\mu_n \geq 0$. This is because a pixel in the image cannot physically have a negative number of decay events (mathematically, the Poisson process is not defined for $\beta < 0$).

Minimization of functions

- The most general techniques for finding the minima of arbitrary functions are termed methods of **optimization**.
- Suppose we are given a function $f(\mathbf{x})$ and are told to find the value of $\mathbf{x}$ at which $f(\mathbf{x})$ has its minimum value.
- A naive approach is to evaluate $f(\mathbf{x})$ over all points in space and choose the point $\hat{\mathbf{x}}$ at which the minimum value occurs.

What are the problems with this approach?

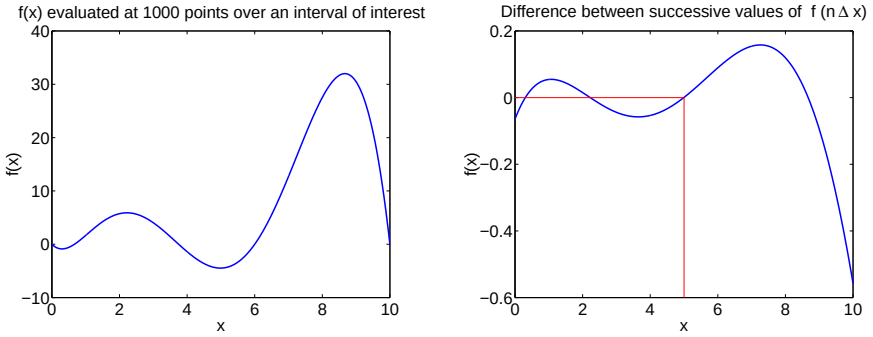
1. _____
2. _____

Under what conditions/modifications could this approach be useful?

1. _____
2. _____
3. _____
4. _____

Minimization of functions

Below, we find the approximate minimum of $f(x)$ over the interval $[0, 10]$ by evaluating the function at 1000 points spaced $\Delta x = 0.01$ apart. The minimum occurs at $x \approx 5$. Note the difference between successive sample values $f(n\Delta x) - f((n-1)\Delta x) = 0$ when $f(x)$ has a turning point.



On a finite interval and given a chosen grid spacing Δx, we can find a solution that is within Δx of the true **global minimum**.

Minimization of functions

- The exhaustive search we just performed is a method of **global optimization**. Global methods are generally applied only to very difficult optimization problems since their computational cost is prohibitive for most applications.
- In two or more dimensions $f(\mathbf{x})$ must be evaluated on a grid of points.
- The higher the dimension of the problem, the larger the set of points at which $f(\mathbf{x})$ must be evaluated to achieve a solution of a certain accuracy. This is called sometimes called the **Curse of Dimensionality**. In our example, we needed a thousand points to get within 0.01 of the true solution. In 3D we would need a billion points.
- Image processing problems usually have a very large number of parameters (pixels or voxels) and so global optimization methods are **seldom of practical value** and are typically used as a **last resort.**

Minimization of functions

- Most optimization algorithms start with an **initial guess** $\mathbf{x}[0]$ of the solution vector and **iteratively refine** this guess until a minimizer $\hat{\mathbf{x}}$ is found.
- The algorithm knows it has found a minimum when, in the process of taking a step towards a lower value of $f(\mathbf{x})$, it encounters a point at which the gradient $\nabla f(\mathbf{x}) = \mathbf{0}$. Practically, we test $\|\nabla f(\mathbf{x})\| < g_{\min} \approx 0$.
- This minimizer $\hat{\mathbf{x}}$ is **not necessarily** the location of the global minimum of the function and is termed a **local minimum**.
- Most local optimization algorithms reduce the value of $f(\mathbf{x})$ at each iteration by taking a step down the surface of $f(\mathbf{x})$ function. $f(\mathbf{x})$ is often called the **objective function** or **cost function**.
- Usually, the gradient of the $f(\mathbf{x})$ plays an important role in determining the **descent direction** along which the algorithm will take its next step.
- The descent direction is described by the unit vector $\mathbf{h}$.

Minimization of functions in 1D

We will now attempt to experience the world through the eyes of an optimization algorithm as it tries to minimize the 1D function $f(x)$ in the previous example.

1. We are given a "black-box" function $f(x)$ that we can evaluate at any value of x that we choose.
2. We are given a "black-box" function $g(x) = \frac{df(x)}{dx}$ that we can evaluate at any value of x that we choose.
3. We are given an initial guess of the solution $x[0] = 7$.

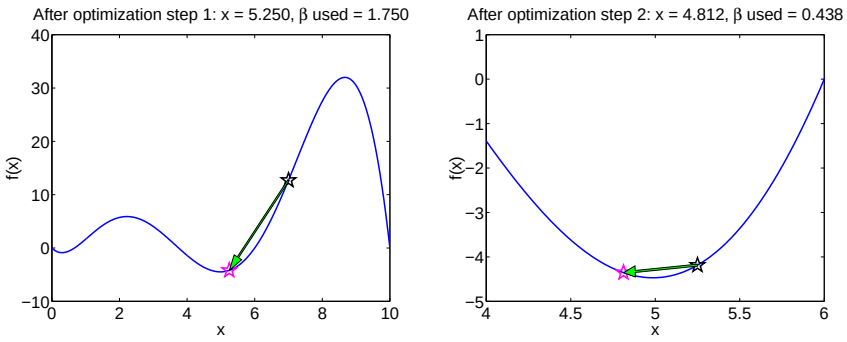
We begin by calling the computer function that calculates $f(x)$. It returns 12.7082. Calling $g(x)$ gives 15.4263. This tells us that $f(x)$ **increases towards the right**. To find a minimum, we must set $h = -1$ to step to the left.

Now we have to "take a step in the dark". How can we work out how far left to go? We must decide on a stepping rule.

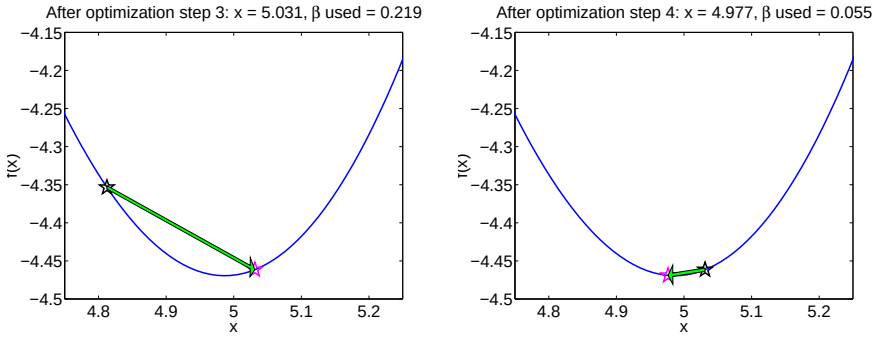
Minimization of functions in 1D

1. We will first try to take a step of arbitrary length $\beta = 3$.
2. We will check if we land at a lower value of the function, in other words whether:
$$f\Big(x[0] + \beta h\Big) < f\Big(x[0]\Big). \tag{1}$$
3. If this is true, we will stay at the new point and set $\lambda = \beta$ and
$$x[1] = x[0] + \lambda h.$$
We then restart the algorithm with $x[1]$ as our best guess of the solution $\hat{x}$ and search for $x[2]$.
4. If our step did not decrease $f(x)$, we divide the current value of β by two, and try taking a shorter step from $x[0]$.
5. We keep reducing β until we find a value of $x[1]$ that satisfies (1). Only then can we proceed to Step 3. If $f(x)$ fails to decrease for $\beta < 0.01$ (arbitary choice of a small β), we accept the current value of x as a local minimum.

Minimization of functions in 1D



Minimization of functions in 1D



After 4 iterations, our simple algorithm finds the minimum of the function as $\hat{x} = 4.977$. This is close to the true solution of $x = 5$.

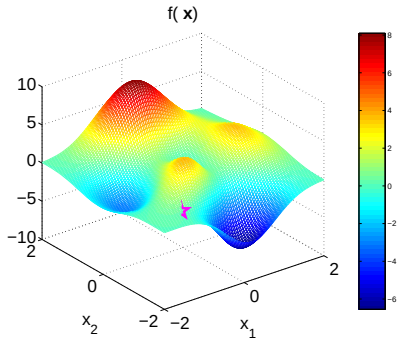
Minimization of functions of arbitrary dimension

- In 1D, an optimization algorithm can choose from **only two** directions along which to take a step.
- In 2+D, the algorithm must choose a single direction from among an **infinite number** of directions.
- The simplest (and greediest) choice is to follow the negative of the gradient of the function towards a local minimum. The algorithm based on this concept is called the method of **steepest descent (SD)**.
- In this course, we study only the SD algorithm, even though algorithms with far more desirable properties exist and are typically used instead of SD in practice.
- Studying SD, however, gives us great insight into local optimization methods, because most algorithms have the same basic recipe:
  1. Pick a direction based on the gradient and/or second derivative of $f(\mathbf{x})$.
  2. Search along this direction for a point at which $f(\mathbf{x})$ is decreased.
  3. Update the current value of $\mathbf{x}$ and try to minimize $f(\mathbf{x})$ further.

Minimization of functions in 2D

Consider the function:

$$\begin{aligned} f(\mathbf{x}) = f(x_1, x_2) \quad = \quad & 3(1-x_1)^2\, \mathrm{e}^{-x_1^2-(x_2+1)^2} \\ & -10\,(x_1/5 - x_1^3 - x_2^5)\, \mathrm{e}^{-x_1^2-x_2^2} \\ & -1/3\, \mathrm{e}^{-(x_1+1)^2-x_2^2} \end{aligned}$$



Minimization of functions of any dimension

By inspection, we see that this function is non-linear and probably does not have a closed-form solution for the minimizer $\hat{\mathbf{x}}$. How can we minimize this 2D function using a local optimization algorithm?

We are given the starting point $\mathbf{x}[0] = [-1.2 \;\; -1.5]^T$.

The following recipe allows us the minimize functions of **any dimension**:

Let $\mathbf{x}[k]$ represent the current iterate:

1. Calculate $f(\mathbf{x}[k])$ and $\mathbf{g}(\mathbf{x}[k]) = \nabla f(\mathbf{x}[k])$. In general:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Minimization of functions of any dimension

2. Check to see if the norm of the gradient $\|\nabla f(\mathbf{x})\|$ is smaller than a lower threshold $g_{\min}$ that we have set. If so, and if the last step was a descent, we are satisfactorily close to a local minimum and can exit the algorithm.
3. Choose a descent direction $\mathbf{h}$. In the SD method, we choose $\mathbf{h} = -\nabla f(\mathbf{x})$.
4. Sample the function in the direction of $\mathbf{h}$ to search for a point for which:
$$f\Big(\mathbf{x}[k] + \beta\mathbf{h}\Big) < f\Big(\mathbf{x}[k]\Big)$$
5. If we can find such a point, set $\lambda = \beta$ and
$$\mathbf{x}[k+1] = \mathbf{x}[k] + \lambda\mathbf{h}$$
6. If we can't find such a point, we must be at a solution and must return it to the user. Otherwise, we set $k = k + 1$ and return to Step 1.

Minimization of functions of any dimension

It is important to note that the search in the direction of $\mathbf{h}$ in Step 4 is **the same as the 1D search** we performed when we minimized the 1D function earlier. Consequently, multidimensional optimization may be viewed as **succesive 1D optimizations along specially chosen direction vectors.** To ensure convergence, we need only ensure that the value of $f(\mathbf{x})$ decreases at **every** step.

We now return to the 2D example. Before we can apply the method of SD, we must derive expressions for the gradient:

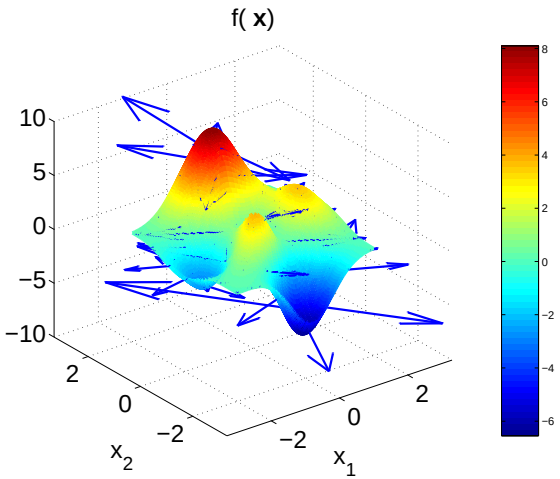
$$\begin{aligned}\frac{\partial f(\mathbf{x})}{\partial x_1} &= -6(1-x_1)\,\mathrm{e}^{-x_1^2-(x_2+1)^2} + \\ &\quad 3(1-x_1)^2\,\mathrm{e}^{-x_1^2-(x_2+1)^2}\,(-2x_1) - \\ &\quad 10\,((1/5 - 3x_1^2)\,\mathrm{e}^{-x_1^2-y^2} + \\ &\quad (x_1/5 - x_1^3 - x_2^5)\,\mathrm{e}^{-x_1^2-x_2^2}\,(-2x_1)) - \\ &\quad 1/3\,\mathrm{e}^{-(x_1+1)^2-x_2^2}(-2(x_1+1))\end{aligned}$$

Minimization of functions of any dimension

$$\begin{aligned}\frac{\partial f(\mathbf{x})}{\partial x_2} &= 3(1-x_1)^2\,\mathrm{e}^{-x_1.^2-(x_2+1)^2}\,(-2(x_2+1)) - \\ &\quad 10\,((-5\,x_2^4)\,\mathrm{e}^{-x_1^2-x_2^2} + \\ &\quad (x_1/5 - x_1.^3 - x_2.^5)\,\mathrm{e}^{-x_1^2-x_2^2}\,(-2x_2)) - \\ &\quad 1/3\,\mathrm{e}^{-(x_1+1)^2-x_2^2}(-2x_2)\end{aligned}$$

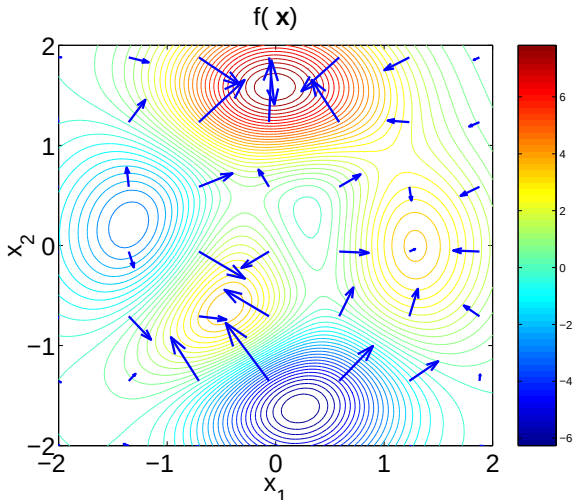
We then set $\mathbf{h} = \mathbf{g}(\mathbf{x})$ and perform the line search.

Minimization of functions in 2D



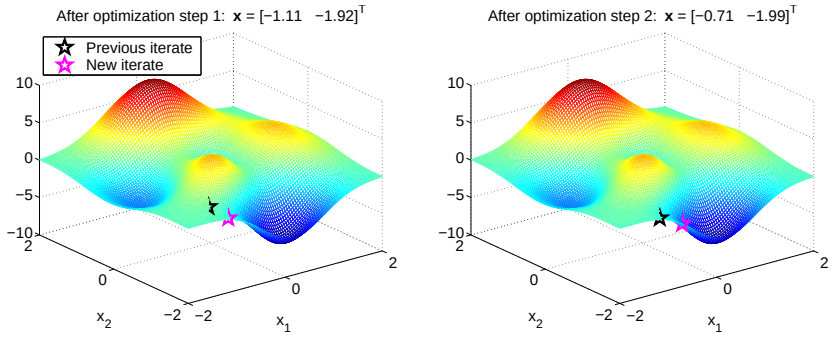
Surface normals are plotted that have length equal to those of the respective gradient vectors. Note how these vectors are long on the steep slopes and short at the turning points.

Minimization of functions in 2D



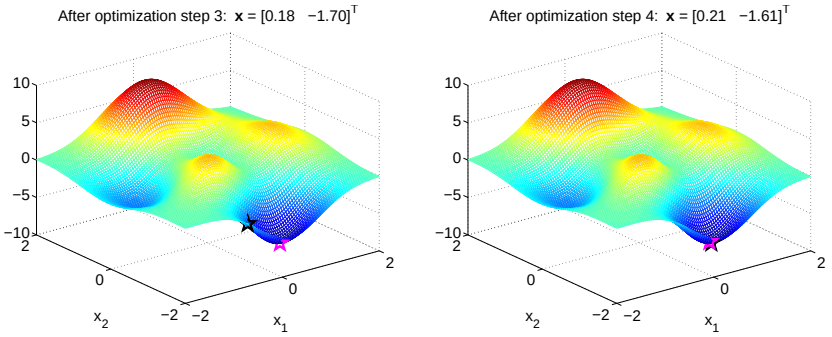
Here, a set of gradient vectors has been plotted over a contour plot of the objective function.

Minimization of functions in 2D



Here, we show the progress of the optimization algorithm. The SD method used utilizes the same crude line search we employed for the 1D example. In this 2D example, $\beta_0 = 0.2$, $\beta_{\min} = 0.035$ and $g_{\min} = 0.75$.

Minimization of functions in 2D



Minimization of functions in 2D

- Our algorithm finds a satisfactory solution after only four iterations.
- However, the initial value of the parameter β must be carefully adjusted to suit the scale of each problem otherwise convergence will be very slow.
- A simple, computationally cheap, and effective step rule is the **Armijo line search**. This algorithm produces a step size $\lambda = \beta^k, k \in \mathcal{Z}$ that satisfies:

$$f\Big(\mathbf{x}[k] + \beta^k \mathbf{h}\Big) - f(\mathbf{x}[k]) \leq \beta^k \alpha \nabla f(\mathbf{x}[k])^T \mathbf{h}$$

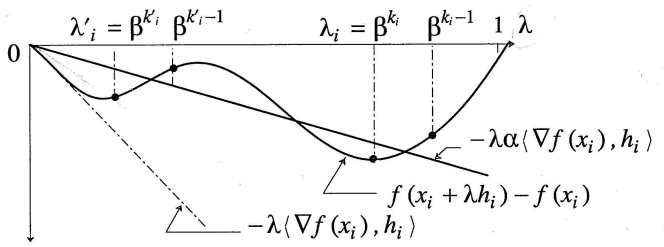
and

$$f\Big(\mathbf{x}[k] + \beta^{k-1} \mathbf{h}\Big) - f(\mathbf{x}[k]) > \beta^{k-1} \alpha \nabla f(\mathbf{x}[k])^T \mathbf{h}$$

This search is much less sensitive to the values of its parameters $\alpha \in (0, 1)$ and $\beta \in (0, 1)$ than our crude line search. α and β control the speed of convergence to the solution. The Armijo method modifies β far more efficiently than our crude method, which simply divides β by two until a decrease in $f(\mathbf{x})$ was achieved. The Armijo method adjusts the step length by raising β to a positive or negative exponent. Thus, steps can grow or shrink. The Armijo method also checks to see that the decrease in the function is sufficient to ensure convergence.

Minimization of functions in 2D

Graphical interpretation of the Armijo line search (from Polak (1997) p.30):



λ'_i and λ_i are both step lengths that satisfy the criteria of the Armijo rule. Whether the former or latter are chosen depends on the values of the parameters α and β. Note that:

$$\langle \nabla f(\mathbf{x}), \mathbf{h} \rangle \equiv \nabla f(\mathbf{x})^T \mathbf{h}.$$

Minimization of functions in 2D

- In the SD algorithm, we have $\mathbf{h} = \mathbf{g}(\mathbf{x})$, so the Armijo criteria become:

  $$f\Big(\mathbf{x}[k] + \beta^k \mathbf{h}\Big) - f(\mathbf{x}[k]) \leq \beta^k \alpha \left\|\nabla f(\mathbf{x}[k])\right\|^2$$

  and

  $$f\Big(\mathbf{x}[k] + \beta^{k-1} \mathbf{h}\Big) - f(\mathbf{x}[k]) > \beta^{k-1} \alpha \left\|\nabla f(\mathbf{x}[k])\right\|^2$$

- We see that, for a step to be acceptable, the decrease in the cost function must exceed some fraction of the length of the gradient vector. This helps protect the algorithm from "getting stuck" as a result of numerical errors present in the gradient and cost function values. These errors are unavoidable in a computer implementation.

Matlab code for an SD algorithm with Armijo line search

```
function [x, f, g, n, xi, fi] = sdarmijo(x0, alpha, beta, n, ...
                                         funcName)

% x0: inital solution estimate
% alpha: convergence parameter
% beta: step size parameter
% n: max. iterations
% funcName: name of user function
% Function must return [f,g] - cost scalar and
% gradient vector

% returns:
% f: final function value
% g: final gradient value
% n: iterations performed
% for postrun progress analysis:
```



```
% xi: collection of x vectors
%     indexed by iteration
% fi: vector of cost function
%     values indexed by iteration

x = x0(:); % make sure x is a column vector
xi{1} = x; % store starting vector in history
k=0;
kMax = 20; % maximum power of beta (smallest
           % possible step
kMin = -50; % largest power of beta (largest
            % possible step

for i = 1:n
  [f, g] = feval(funcName, x); % evaluate cost
                               % function and gradient
  fi(i) = f; % record history of cost
  h = -g; % set search direction to steepest descent direction
```

```
% Armijo search loops
if feval(funcName, x + beta^k*h) - f > beta^k*alpha*g.'*h
  while feval(funcName, x + beta^k*h) - f > ...
          beta^k*alpha*g.'*h
    k=k+1; % take smaller step because current step
           % increases function value
    if k > kMax
      break % step is very small, we must be done
    end %if
  end %while
else
  while feval(funcName, x + beta^k*h) - f <= ...
          beta^k*alpha*g.'*h
    k=k-1; % take bigger step because we have reduced the
           % function value, but might do better
   % with a larger step
    if k < kMin
```



```
      break % step is getting too large
    end %if
  end %while
  k=k+1;
end %if

% update solution
x = x + beta^k*h;
% update history
xi{i+1} = x;
end %for

% update cost function history and
% find final gradient vector
[fi(i+1), g] = feval(funcName, x);
```


Matlab code for demo 2D cost function to be minimized

```
function [f,g] = peaksfn(X,Y)

% prepares "peaks" function for 2D optimization demo
% allow function to be called with a vector argument [x y].'
if nargin == 1
  Y = X(2);
  X = X(1);
end

% cost function
f = 3*(1-X).^2.*exp(-(X.^2) - (Y+1).^2) ...
   - 10*(X/5 - X.^3 - Y.^5).*exp(-X.^2-Y.^2) ...
   - 1/3*exp(-(X+1).^2 - Y.^2) ;

% partial with respect to x1
gx = -6*(1-X).*exp(-(X.^2)-(Y+1).^2) + 3*(1-X).^2 .* ...
```



```
   exp(-(X.^2) - (Y+1).^2).*(-2*X) - 10 * ...
    ( (1/5 - 3*X.^2).*exp(-X.^2-Y.^2) + ...
      (X/5 - X.^3 - Y.^5).*exp(-X.^2-Y.^2).*(-2*X) ) - 1/3 * ...
   exp(-(X+1).^2 - Y.^2).*(-2*(X+1));

% partial with respect to x2

gy = 3*(1-X).^2.*exp(-(X.^2) - (Y+1).^2).*(-2*(Y+1)) - 10 * ...
    ( (- 5*Y.^4).*exp(-X.^2-Y.^2) + ...
      (X/5 - X.^3 - Y.^5).*exp(-X.^2-Y.^2).*(-2*Y) ) - 1/3 * ...
   exp(-(X+1).^2 - Y.^2).*(- 2*Y);


g = [gx; gy];
```

Matlab code for invoking optimization algorithm

```
>> [x,f,g, n, xi, fi] = sdarmijo([-1.2 -1.5], 0.5, 0.5, ...
                                  10, 'peaksfn');

fi =
  Columns 1 through 7

0.3776  -0.3786  -2.8445  -6.3106  -6.5447  -6.5499  -6.5509

  Columns 8 through 11
-6.5511  -6.5511  -6.5511  -6.5511

>> [xi{1:10}]
ans =

  Columns 1 through 7
-1.2000  -1.0917   0.8894   0.2435   0.2522   0.2392   0.2331
```



```
-1.5000  -2.0238  -1.9354  -1.4963  -1.6138  -1.6216  -1.6239

  Columns 8 through 10
0.2304    0.2292    0.2287
-1.6248   -1.6252   -1.6254

>> norm(g)

ans =

    0.0034
```